TEMA 8: ENTORNO DE EJECUCIÓN (RUNTIME).

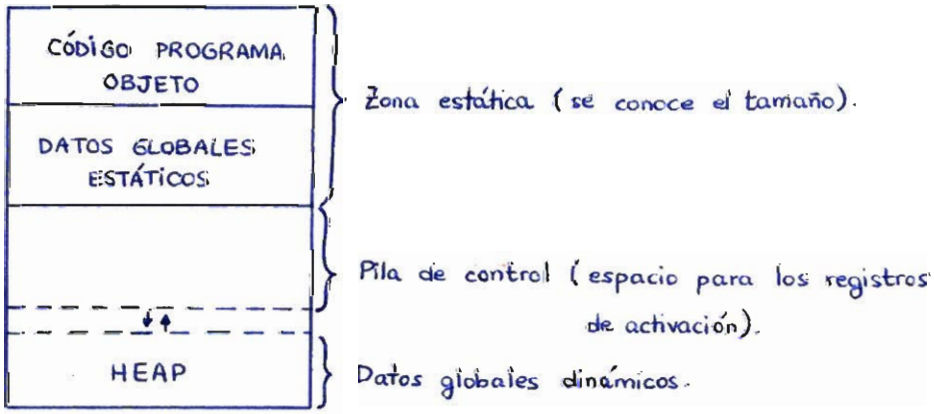
El entorno de ejecución viene dado por la estructura de memoria (incluidos los registros de la unidad central de proceso - CPU -) y la forma de gestionarla, de manera que permita desarrollar adecuadamente el proceso de ejecución de un programa.

Tiempo de compilación ≡ Tiempo de ejecución del compilador.
Tiempo de ejecución del compilador ≠ Tiempo de ejecución del programa compilado.

1. ORGANIZACIÓN DE LA MEMORIA EN TIEMPO DE EJECUCIÓN.

La organización de la memoria es necesaria para construir el código objeto de forma correcta.

El S.O. asigna un espacio de memoria al programa, que se estructura de la siguiente manera:



El código del programa se coloca al inicio de la memoria asignada.

El código fuente del programa estará compuesto por distintos bloques o procedimientos. Los procedimientos tendrán variables propias (locales) y para que se puedan ejecutar deberán tener un espacio para almacenarlas. Este espacio propio de cada bloque se denomina **REGISTRO DE ACTIVACIÓN.**

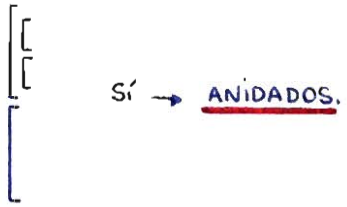
Activación de procedimientos:
- Funciones.
- Procedimientos.
- Programa principal.

* SUPOSICIONES DE CONTROL:

Determinación de qué características permitimos y cuáles no en la ejecución de un procedimiento:

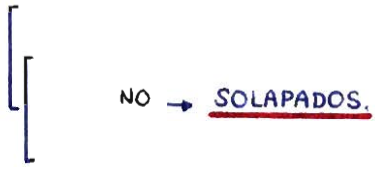
* Control secuencial.

* Procedimientos anidados se permiten, pero no procedimientos solapados.



Sí → ANIDADOS.

El procedimiento llamado tiene que terminar antes que el invocador.



NO → SOLAPADOS.

* Se permite la recursividad directa (P → P) y la mutua. (P → Q → P).

Cada ejecución de un procedimiento se denomina ACTIVACIÓN del procedimiento. La duración de una activación de un procedimiento es el tiempo desde que comienza hasta que termina su ejecución, incluyendo el tiempo que tardan los procedimientos que él llama.

Si A y B son activaciones de procedimientos, entonces sus duraciones o bien no se solapan o están anidadas, es decir, si se entra a B antes de salir de A, entonces el control debe abandonar B antes de abandonar A.

Un procedimiento es recursivo si puede comenzar una nueva activación antes de que haya terminado una activación anterior del mismo procedimiento. Por tanto, si el procedimiento es recursivo, pueden estar funcionando varias de sus activaciones al mismo tiempo. Un procedimiento recursivo no se llama a sí mismo necesariamente, sino que puede llamar a otro que acabe llamándole a él en algún momento.

* ÁRBOL DE ACTIVACIÓN:

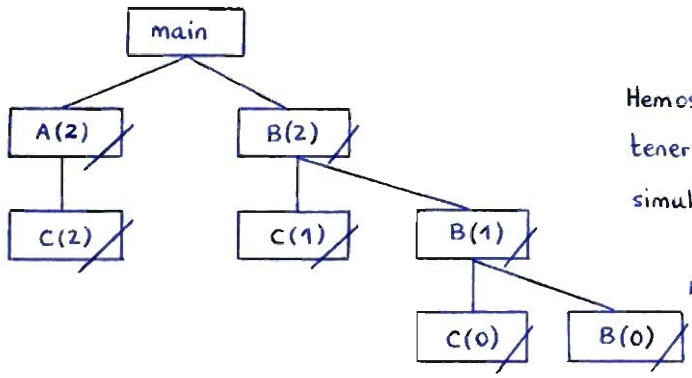
Se puede utilizar un árbol, llamado árbol de activación, para representar la forma en que el control entra y sale de las activaciones.

Un árbol de activaciones tiene las siguientes características:

- Cada nodo representa una activación de un procedimiento.
- La raíz representa la activación del programa principal.
- El nodo para A es el padre del nodo para B si y sólo si el control fluye de la activación A a la B.
- El nodo para A está a la izquierda del nodo para B si y sólo si la duración de A ocurre antes que la duración de B, es decir, si se ejecuta A antes que B.

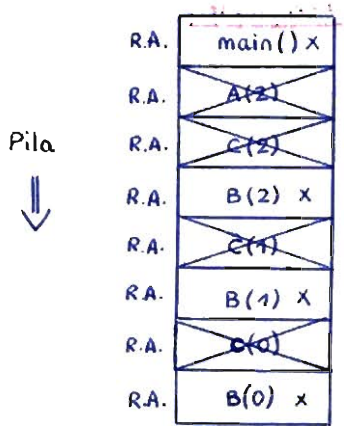
Ejemplo:

```
main () {
    int m = 2;
    A(m);
    B(m); }

A (int j) {
    c(j); }

B (int j) {
    int b;
    if (i ≠ 0) {
        b = i - 1;
        c(b);
        B(b); } }

C (int i) { ... }
```



Hemos llegado a tener 4 activaciones simultáneas.

Como no hay entrada de datos, en tiempo de compilación sabemos ya todas las activaciones que se producirán:



2. ESTRATEGIAS DE ASIGNACIÓN DE MEMORIA.

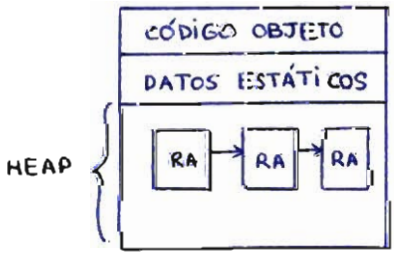
Existen diferentes estrategias de asignación de memoria para los registros de activación (RA):

1) ASIGNACIÓN ESTÁTICA: Sólo es válida para lenguajes que no admiten recursividad ni datos dinámicos, como Fortran. A cada programa siempre se le asigna la misma zona de memoria para su registro de activación.

2) ASIGNACIÓN DINÁMICA DE PILA: La pila crece hacia abajo. Con esta estrategia el último procedimiento en ser llamado es el primero que tiene que acabar. Permite recursividad y datos dinámicos, pero no es válido para programación concurrente.

=> Los datos dinámicos se pueden situar en la pila o en el heap.

3) ASIGNACIÓN DINÁMICA POR HEAP: Asigna los RA buscando huecos en la zona de memoria asignada. Es menos restrictivo y sirve para cualquier paradigma de programación.



Complica la gestión de memoria.

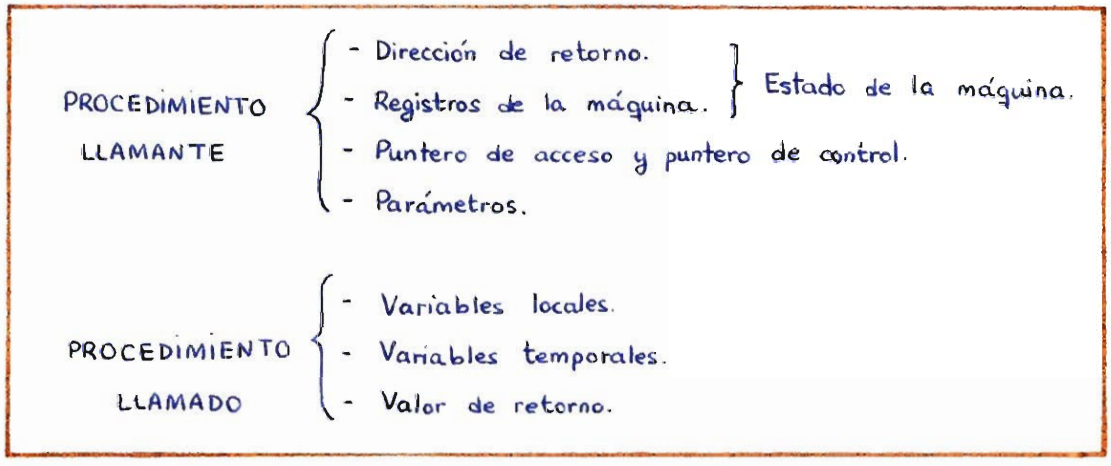
3. REGISTRO DE ACTIVACIÓN.

El registro de activación (RA) es un conjunto de posiciones contiguas que se asignan a un procedimiento para poder llevar a cabo la ejecución del mismo. Cada procedimiento tendrá un registro de activación del tamaño que necesite. Este tamaño se conoce en tiempo de compilación.

Contenido del R.A:

- Dirección de retorno.
- Estado de la máquina.
- Parámetros.
- Variables locales.
- Variables temporales.
- Valor de retorno.
- Puntero de acceso.
- Puntero de control.

El registro de activación tiene un orden predefinido. Parte de la información del RA la coloca el procedimiento que hace la llamada (por ejemplo, dirección de retorno, parámetros actuales,...), mientras que otra parte la colocará el procedimiento llamado.

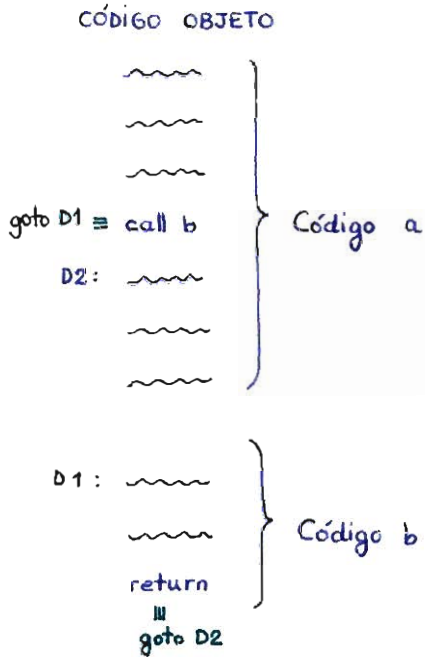


También se actualizará el valor de la cima de la pila:

SP + Tamaño_RA_Llamante

3.1. DIRECCIÓN DE RETORNO.

El procedimiento llamante debe almacenar en el RA la dirección de retorno para que, al finalizar el llamado, pueda devolver el control al procedimiento anterior.



En el RA de "b" tendré D2 en la dirección de retorno para que cuando "b" termine sepa dónde está "a" y en qué instrucción se quedó.

3.2. ESTADO DE LA MÁQUINA.

Espacio donde salvar el contenido de los registros de la máquina. Estos registros deben reponerse cuando el control vuelva al procedimiento.

3.3. PARÁMETROS.

Este campo es utilizado por el llamante para proporcionarle los parámetros actuales al procedimiento que recibe la llamada.

Hay 3 modos de paso de parámetros:

- Referencia.
- Valor.
- Copia / Restauración.

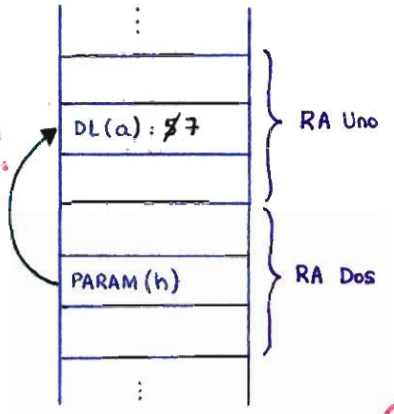
* PASO POR REFERENCIA:

El procedimiento llamante pasa al llamado la dirección del parámetro, no el valor.

```
proc Uno
    var a;

    proc Dos (h)
        h := 7;

    a := 5;
    Dos (a)
```

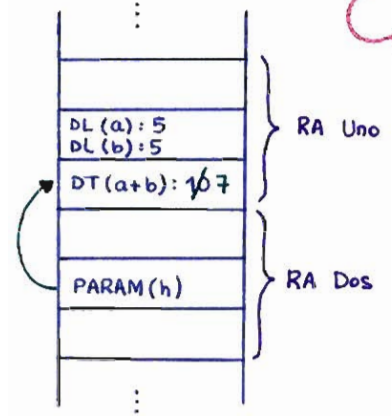
Parámetros formales.

Parámetros actuales.



DL ≡ Dato local.
DT ≡ Dato temporal.

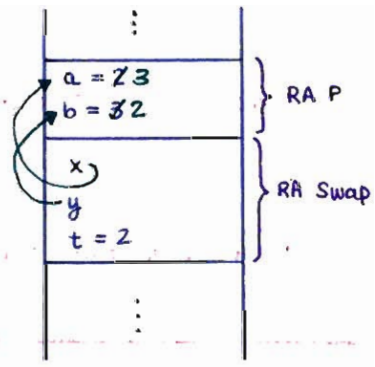
```
proc Uno
    var a, b;

    proc Dos (h)
        h := 7;

    a := 5 ; b := 5;
    Dos ( a + b);
```



¿Cómo resuelvo esto? ¿Qué hago con (a + b)? Calculo (a+b) en un dato temporal y lo que paso por referencia es la dirección del valor temporal. Sin embargo, esto es absurdo porque cuando Dos devuelva el control a Uno habré perdido ese valor.

```
proc Swap ( x, y : int)
    var t: int;
begin
    t := x ;
    x := y;
    y := t;
end
```

```
proc P
    var a, b : int;
begin
    a := 2 ;
    b := 3;
    Swap (a, b);
end
```

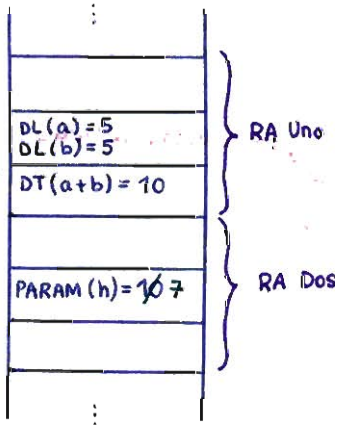


* PASO POR VALOR :

El procedimiento llamante pasa al llamado el valor del parámetro.

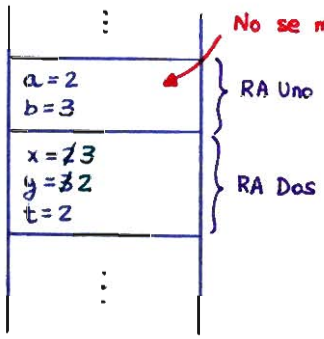
```
proc Uno
   var a, b;

   [ proc Dos (h)
        h := 7;

   a := 5; b := 5;
   Dos (a+b);
```

No se modifica su valor.




```
proc Swap ( x, y : int)
    var t : int;
begin
    t := x;
    x := y;
    y := t;
end
```

```
proc P
   var a, b : int;
begin
   a := 2;
   b := 3;
   Swap (a,b);
end
```



Param a
Param b
Call Swap

Es importante distinguir el paso de parámetros por valor y por referencia porque generan código final diferente :

- REFERENCIA => Paso una dirección => Cambio en el RA del llamante.
- VALOR => Paso valores concretos => Cambio en el RA del llamado.

Independientemente del paso de parámetros que se utilice, hay que reservar espacio en el RA. Es frecuente que para una mayor eficiencia los parámetros se pasen en los registros de la máquina. Aún así, tiene que existir espacio para los parámetros en el registro de activación.

* COPIA / RESTAURACIÓN :

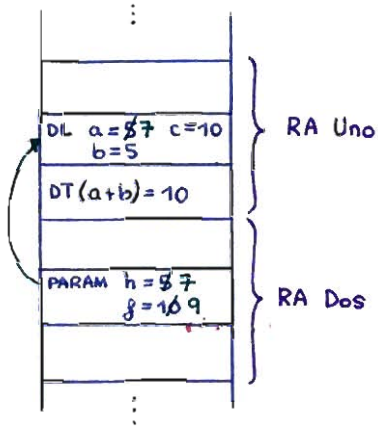
Se evalúan los parámetros actuales. Para los que tienen una dirección, se pasa su dirección (paso por referencia) y su valor (paso por valor). Para los que no tengan dirección, se pasa su valor.

Datos temporales

Se trabaja sobre el registro de activación del procedimiento llamado. Cuando se devuelve el control se restaura el valor de los que sí tenían dirección.

Ejemplo:

```
proc Uno
    var a, b, c;

    [ proc Dos (h, g)
          h := 7;
          g := 9;

    a := 5; b := 5; c := a + b;
    Dos (a, a+b);
```



- Cuando Uno llama a Dos le pasa:
  - * Dirección y valor de "a".
  - * Valor de "a+b".

- Se ejecuta Dos, que modifica en su RA los valores de los parámetros.

- Cuando termina Dos y devuelve el control a Uno, actualizará el valor de "a" en el RA de Uno y el valor temporal de "a+b=9" se perderá.

3.4. VALOR DE RETORNO.

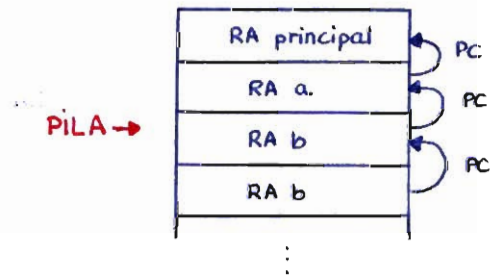
Si el procedimiento llamado es una función, devolverá algún valor. Éste se almacenará en el espacio del RA reservado para tal fin.

3.5. PUNTERO DE CONTROL (PC).

Se trata de un puntero que se establece desde el registro de activación de un procedimiento al registro de activación del que lo ha llamado.

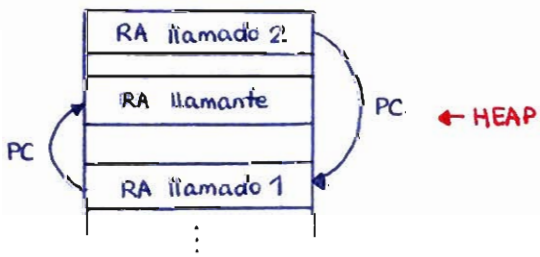
⇒ Encadena el llamado con el llamante.

Cuando se utiliza asignación dinámica por pila el puntero de control no es necesario porque ya se sabe quién ha llamado a quien: el procedimiento llamante se sitúa en la pila en la posición inmediatamente anterior al llamado. Este puntero es necesario únicamente cuando se utiliza asignación de memoria dinámica por heap.



Si en nuestro lenguaje el procedimiento llamado acaba antes que el llamante, es decir, si existen los procedimientos anidados, se utiliza asignación dinámica de la memoria por pila. Por tanto, el puntero de control (PC) no es necesario.

Si en nuestro lenguaje el procedimiento llamante puede acabar antes que el llamado, es decir, si existen los procedimientos solapados, se debe utilizar asignación dinámica de la memoria por heap. Por tanto, aquí sí será necesario el puntero de control.



3.6. PUNTERO DE ACCESO (PA)

El puntero de acceso permite acceder a las variables no locales.

```
Procedure A
    Var a;  ← Variable no local.
    Procedure B
        Var b;
        a = 8;
```

Cuando el lenguaje no tiene estructura de anidamiento de bloques, el puntero de acceso no tiene sentido:

* Ámbito léxico sin procedimientos anidados (C) → Tendremos:
  - Variables ESTÁTICAS: Se almacenan en la zona de datos estáticos del programa.
  - Variables LOCALES: Se almacenan en el RA actual.

  Por tanto, no se necesita puntero de acceso (PA).

* Ámbito léxico con procedimientos anidados (Ada, Pascal) → Tendremos:
  - Variables GLOBALES: Se almacenan en la zona de datos estáticos del programa.
  - Variables LOCALES: Se almacenan en el RA actual.
  - Variables NO LOCALES: Se encuentran almacenadas en un RA que no es el actual.

  Por tanto, se necesita puntero de acceso (PA).

El puntero de acceso de un registro de activación apunta a la activación más reciente de su bloque padre.

Bloque padre ≠ Procedimiento que le llama.

Para acceder a las variables no locales, en tiempo de ejecución no se busca nada, ya se sabe exáctamente en qué registro de activación se encuentra cada una de ellas. Esta búsqueda se hace en tiempo de compilación al hacer las tablas de símbolos. En tiempo de ejecución ya sé, por ejemplo, que la variable "x" está en el RA padre (1 puntero de acceso) y la variable "y" en el RA abuelo (2 punteros de acceso).

* ESTABLECIMIENTO DEL PUNTERO DE ACCESO:

El procedimiento que hace la llamada es el responsable de colocar el puntero de acceso del procedimiento llamado. Si proc. "Uno" llama a proc. "Dos", entonces "Uno" tendrá que colocar el puntero de acceso de "Dos" a la activación más reciente de "Uno".

Un concepto importante es el de **PROFUNDIDAD DE ANIDAMIENTO** o profundidad de bloque:

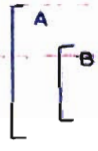
- Si un padre tiene profundidad "x", su hijo tendrá profundidad "x+1".
- El programa principal tiene profundidad 1.

El ámbito de definición de variables, es decir, las profundidades, se conoce en tiempo de compilación, mirando el programa fuente.

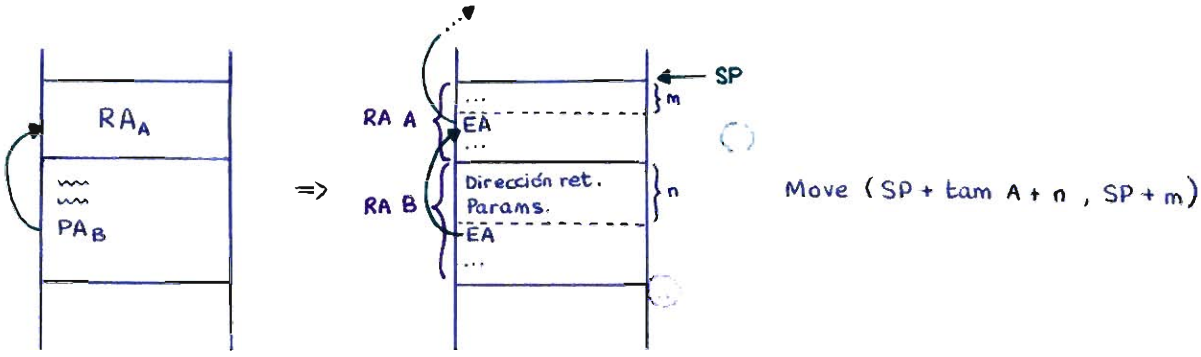
Si un procedimiento A de profundidad "x" ($prof_A$) llama a un procedimiento B de profundidad "y" ($prof_B$), pueden ocurrir 2 casos:

1.- A llama a B, $prof_A < prof_B$:

A es el padre de B (A llama a un hijo directo):



En este caso colocar el puntero de acceso de B es sencillo porque A sabe dónde está el registro de activación del padre de B: A es el padre, el puntero de acceso de B debe apuntar al registro de activación de A.

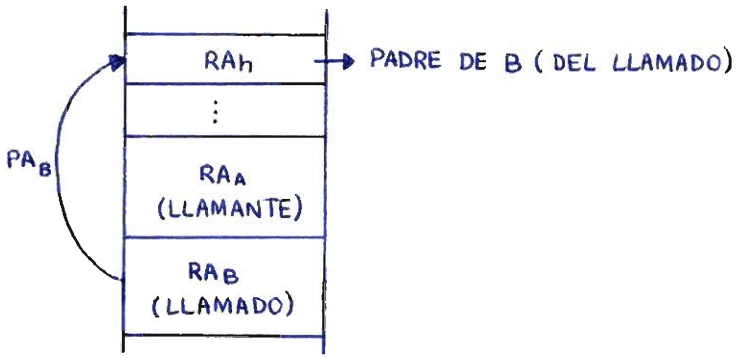


2.- A llama a B, $prof_A \geq prof_B$:

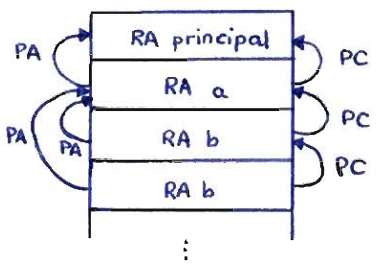
- A llama a su padre, abuelo, etc => $prof_A > prof_B$.
- A llama a un hermano o a sí mismo recursivamente => $prof_A = prof_B$.
  En este segundo caso, el padre de A y B es el mismo.

El puntero de acceso del RA del procedimiento llamado (B) se establece como la distancia (número de saltos) desde el puntero de acceso del llamante (A) hasta el registro de activación más reciente del padre del llamado (B):

$prof_A - prof_B + 1$ saltos

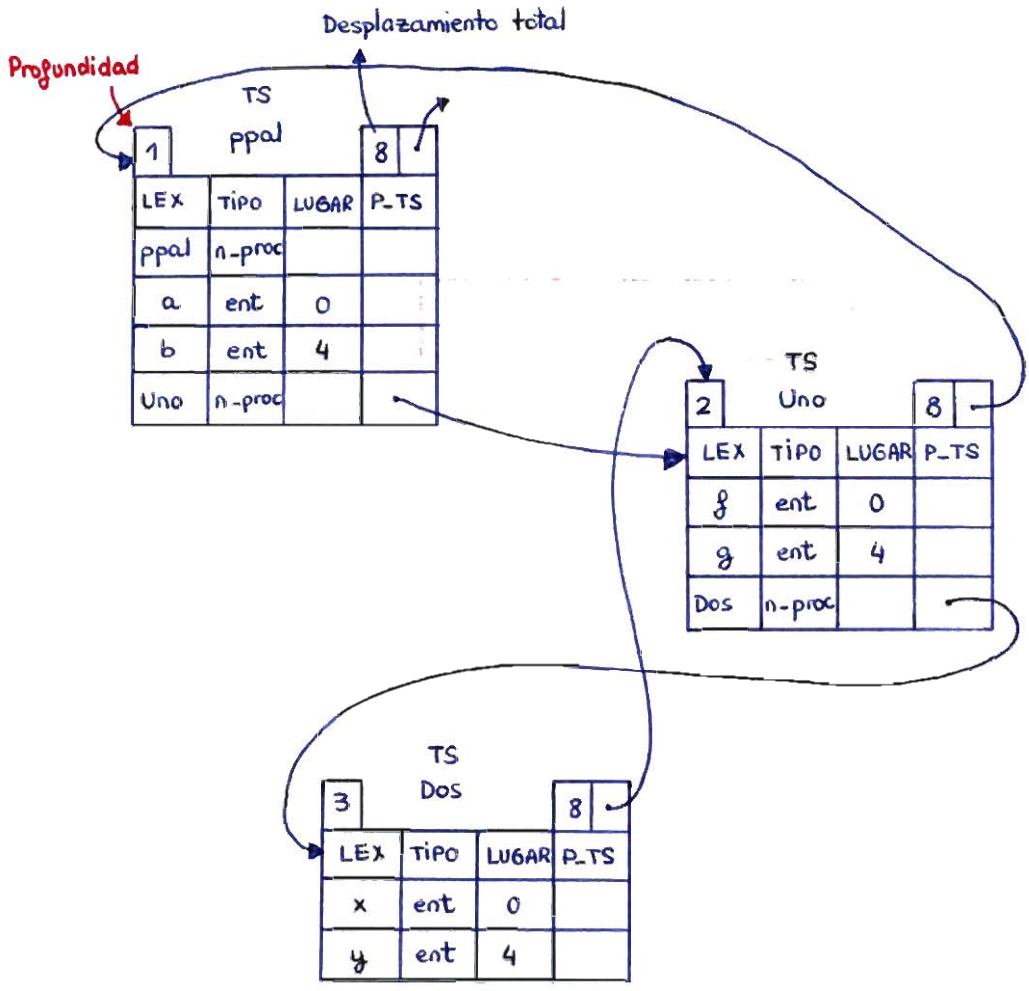


Ejemplo:



Ejemplo: Tablas de símbolos y ámbito de definición de variables.

```
program ppal (1)
  var a, b : integer

  procedure Uno (2)
    var f, g : integer

    procedure Dos (3)
      var x, y : integer

      x := 7 + f + a
```



* ACCESO A VARIABLES NO LOCALES:

Para acceder a las variables no locales se utiliza el puntero de acceso. La dirección de la variable no local tiene 2 componentes:

* Número de punteros de acceso que hay que recorrer para acceder a la variable (número de saltos) → Diferencia entre la profundidad del bloque que usa la variable ($prof_{USO}$) y la profundidad del bloque en el que está definida ($prof_{DEF}$):

$$prof_{USO} - prof_{DEF}$$

* Desplazamiento de la variable dentro del registro de activación en el que está definida ($despl_{VARIABLE}$).

Con estas dos componentes se puede localizar perfectamente la variable no local.

IMPORTANTE:

- Establecer el puntero de acceso (PA) por parte del procedimiento llamante:

$$prof_A - prof_B + 1 \text{ saltos} \quad \begin{matrix} A = \text{llamante} \\ B = \text{llamado} \end{matrix}$$

- Determinar la dirección de una variable no local:

$$(prof_{USO} - prof_{DEF}\ ,\ despl_{VARIABLE})$$

PUNTERO DE CONTROL: INNECESARIO SI SE USA PILA.
NECESARIO SI HAY DATOS DINÁMICOS (HEAP).

PUNTERO DE ACCESO: NECESARIO SÓLO SI SE PERMITE ANIDAMIENTO.

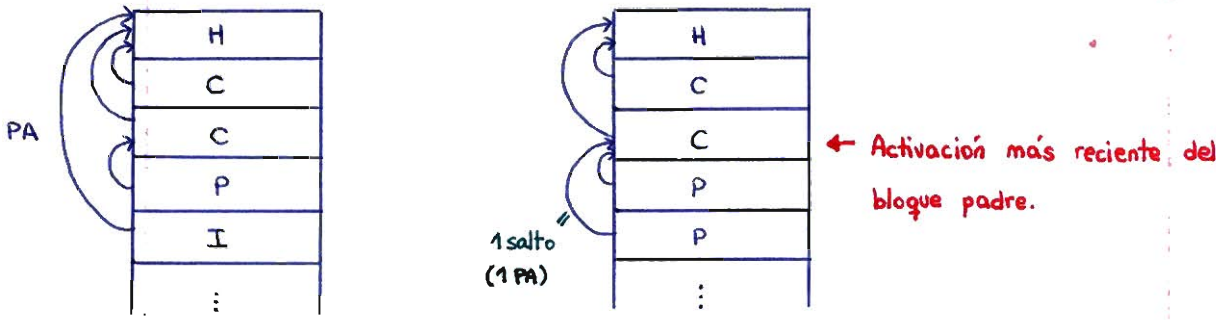
Ejemplo:

```
proc H ①
    var a, x;
    proc L ②
        var i;
        usa a;
        proc M ③
            ...
    proc I ②
        var i, j;
        usa x, a, i, j;
    proc C ②
        var w, v, k, u;
        proc P ③
            var b, i, o;
            usa a, v;
```

v → procedimiento C.

a → procedimiento H.

C no puede llamar a M.



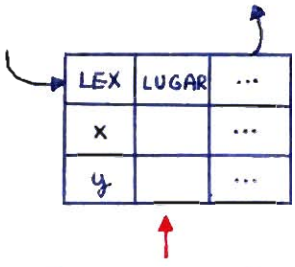
- Un procedimiento llama a un hermano => I llama a L => 2 - 2 + 1 = 1 salto (desde I)
  (prof_I, prof_L)

- Un procedimiento se llama a sí mismo (recursividad directa) => P llama a P =>
  => 3 - 3 + 1 = 1 salto (desde P).

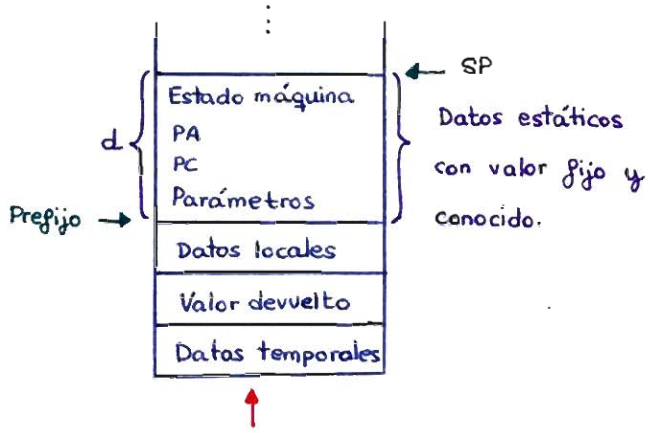
- Un procedimiento llama a su hijo => C llama a P => 2 - 3 + 1 = 0 saltos (desde C).

3.7. VARIABLES LOCALES.

Las variables locales se encuentran almacenadas en el registro de activación de su procedimiento SIEMPRE en la misma secuencia (orden) que en la tabla de símbolos de su procedimiento.



TIEMPO DE COMPILACIÓN



TIEMPO DE EJECUCIÓN

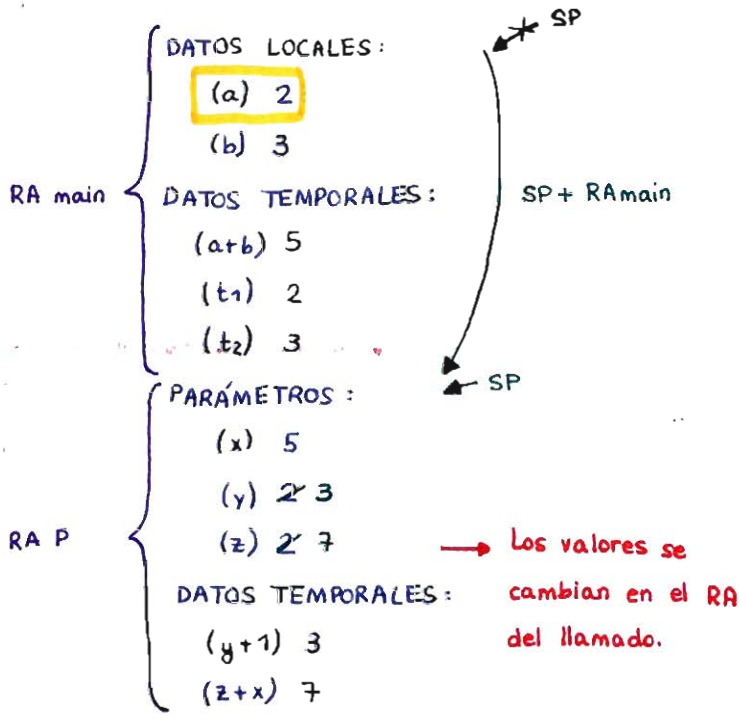
Acceso a las variables locales → SP + d + despl VARIABLE

SP + d: prefijo

despl VARIABLE: campo "lugar" de la tabla de símbolos.

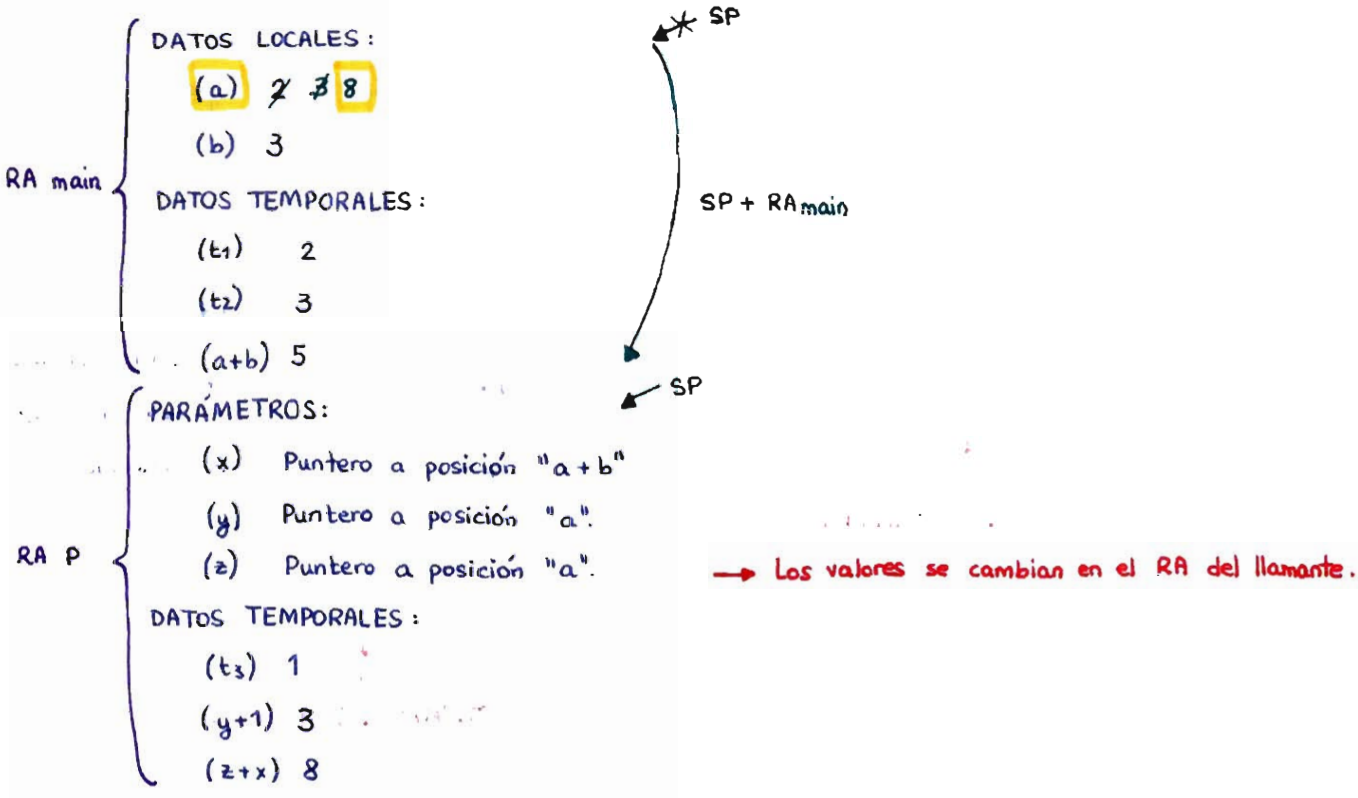
Ejemplo: Paso de parámetros por valor.

```
PROGRAM main();

    PROCEDURE P(x,y,z)
    BEGIN
        y:=y+1;
        z:=z+x;
    END

BEGIN
    a:=2;
    b:=3;
    P(a+b, a, a);
    print a;
END
```



→ En este caso, print imprime "2".

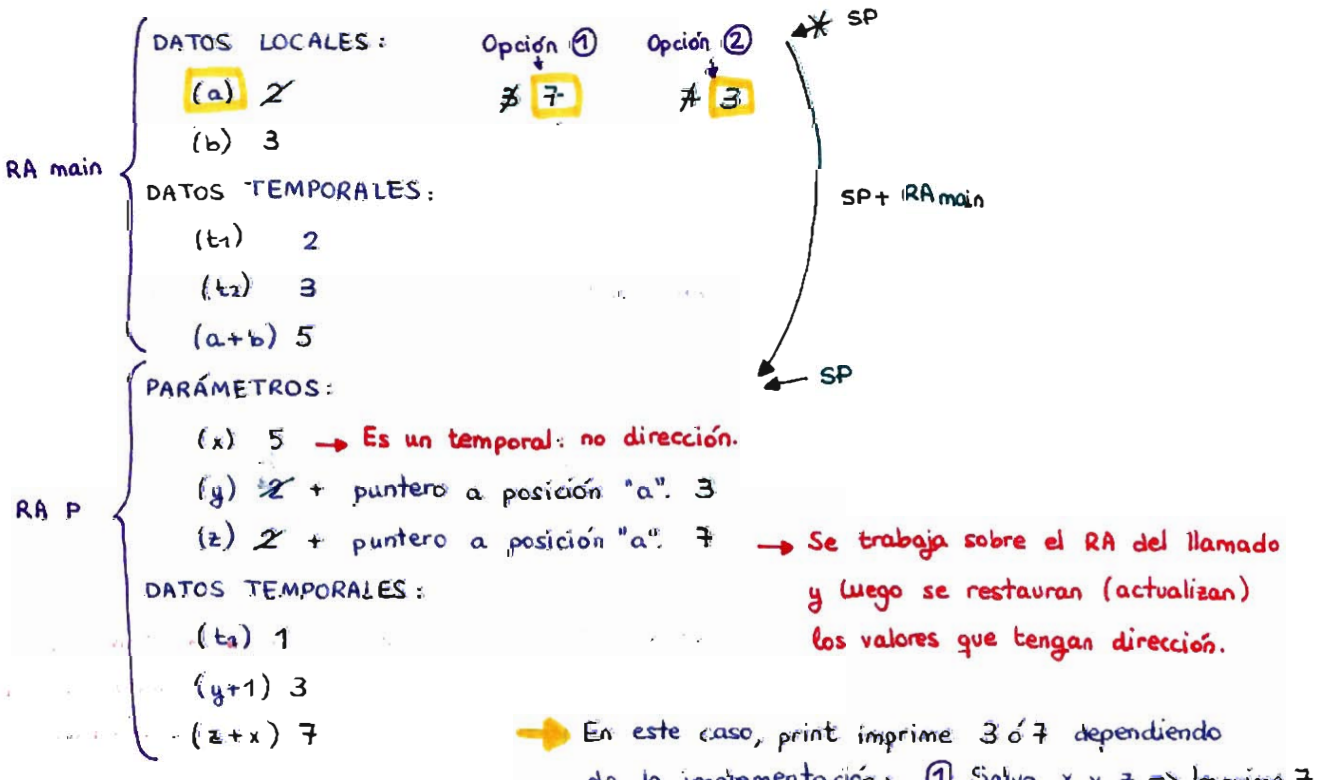
Ejemplo: Ejemplo anterior, ahora con paso de parámetros por referencia.



→ Los valores se cambian en el RA del llamante.

→ En este caso, print imprime "8".

Ejemplo: Ejemplo anterior, ahora con paso de parámetros por copia/restauración.



→ Es un temporal: no dirección.

→ Se trabaja sobre el RA del llamado y luego se restauran (actualizan) los valores que tengan dirección.

→ En este caso, print imprime 3 ó 7 dependiendo de la implementación: ① Salvo x, y, z => Imprime 7.
② Salvo z, y, x => Imprime 3.

Orden de restauración.

* NOTA.- 1 nombre ⟶✗⟶ 1 espacio de memoria.

```
program a();
    procedure b(...);
        var a;
        var b;
    begin
        ...
    end
begin
    b(...);
end
```

- Para el caso de "a":
  - La primera declaración de "a" se asigna como nombre de un programa completo.
  - La segunda "a" se declara como una variable del tipo que sea.

- Para el caso de "b":
  - La primera "b" se declara como un subprograma (procedimiento).
  - La segunda "b" se declara como una variable del tipo que sea.

COMPILADORES. Septiembre 2001

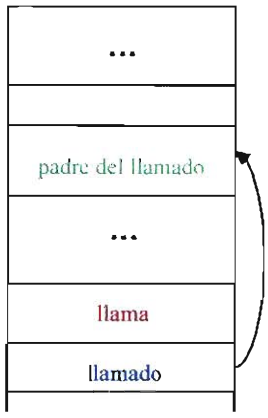
UNA INDICACIÓN A LA SOLUCIÓN DEL EJERCICIO 3

Se está preguntando a qué registro de activación (en adelante, RA) ha de apuntar el nuevo puntero de acceso (en adelante, PA) y cómo se sabe dónde está ese registro.

Por lo tanto, las dos respuestas que el alumno debe dejar claras son:

1. Por tratarse de un lenguaje con **ámbito léxico y procedimientos anidados** es seguro que el PA del RA del llamado **ha de apuntar siempre al RA** más reciente del procedimiento **que es** padre del llamado.

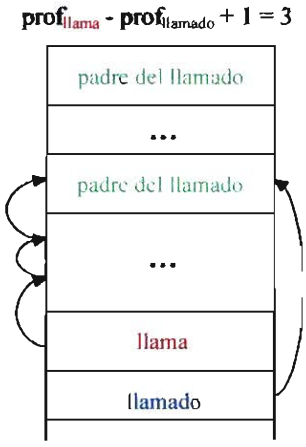
2. El RA al que ha de apuntar el PA del llamado es el que está "(profundidad $_{llama}$ - profundidad $_{llamado}$ + 1) PA's más allá" que el RA del que hace la llamada



Todo esto se explica en detalle a continuación.

La dificultad a resolver es cómo sabe el procedimiento que llama calcular en qué posición de la pila está el RA correspondiente a la activación más reciente del procedimiento padre del que acaba de ser llamado (la cuestión es que no podemos conocer el contenido de la pila de RA's mientras no se ejecute el programa y, sin embargo, tenemos que saber en tiempo de compilación "dónde está exactamente" el RA más reciente del procedimiento padre).

La existencia de **anidamiento** nos permite definir el concepto de **profundidad** de un procedimiento, que es lo que se utiliza para localizar la ubicación en memoria del RA al que se ha de apuntar.



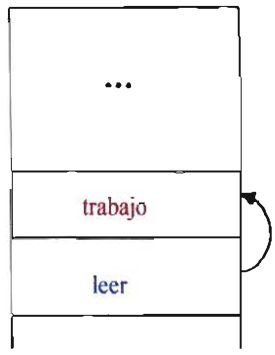
El procedimiento que hace la llamada es el que **se encarga de reservar espacio para el RA del procedimiento** llamado, y de dar valor al campo para PA (y a algunos otros campos). Para establecer el PA utiliza la fórmula

$$\textbf{profundidad}_{llama} - \textbf{profundidad}_{llamado} + 1$$

que le da el número de PA's que se han de recorrer desde su propio RA para alcanzar el RA al que tiene que apuntar el PA del RA que se está creando (y que corresponde al procedimiento llamado).

Se comprueba a continuación que esto funciona para cada uno de los posibles casos de llamadas de este ejercicio.

Cuando **un procedimiento llama a su hijo**, está claro que el PA del RA del llamado ha de apuntar al RA del que ha hecho la llamada. Como la profundidad del hijo es uno más que la del padre, la fórmula de las profundidades para este caso nos da:

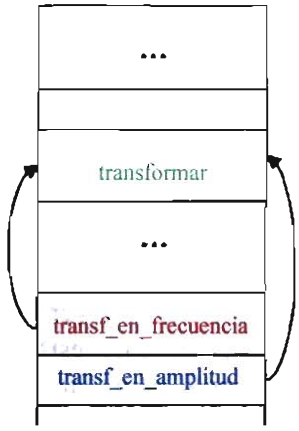


$$\text{profundidad}_{\text{llama}} - \text{profundidad}_{\text{llamado}} + 1 =$$

$$x - (x+1) + 1 = 0$$

lo cual quiere decir que el RA al que hay que apuntar es justo el del procedimiento que ha hecho la llamada.

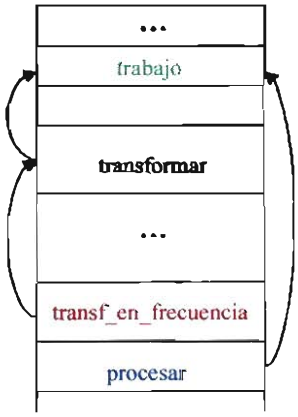
Cuando **un procedimiento se llama a sí mismo, o llama a uno de igual profundidad** que él, el que llama y el llamado tienen el mismo padre. Ello quiere decir que el PA del RA del llamado ha de apuntar al mismo RA al que apunta el PA del que ha hecho la llamada. Es decir, recorriendo un PA desde el RA del que hace la llamada alcanzamos el RA más reciente del padre del llamado, y ese mismo resultado nos lo da la fórmula:



$$\text{profundidad}_{\text{llama}} - \text{profundidad}_{\text{llamado}} + 1 =$$

$$x - x + 1 = 1$$

El último caso es el de **un procedimiento que llama a uno de menor profundidad que él**. En esta situación es seguro que el padre del procedimiento llamado **engloba también** al procedimiento que ha hecho la llamada, por lo que recorriendo PA's se llega desde el RA del que hace la llamada hasta el RA que interesa. El número exacto de PA's a recorrer viene dado por:



$$\text{profundidad}_{\text{llama}} - \text{profundidad}_{\text{llamado}} + 1$$